redux vs alt

@uldissturms

objectives

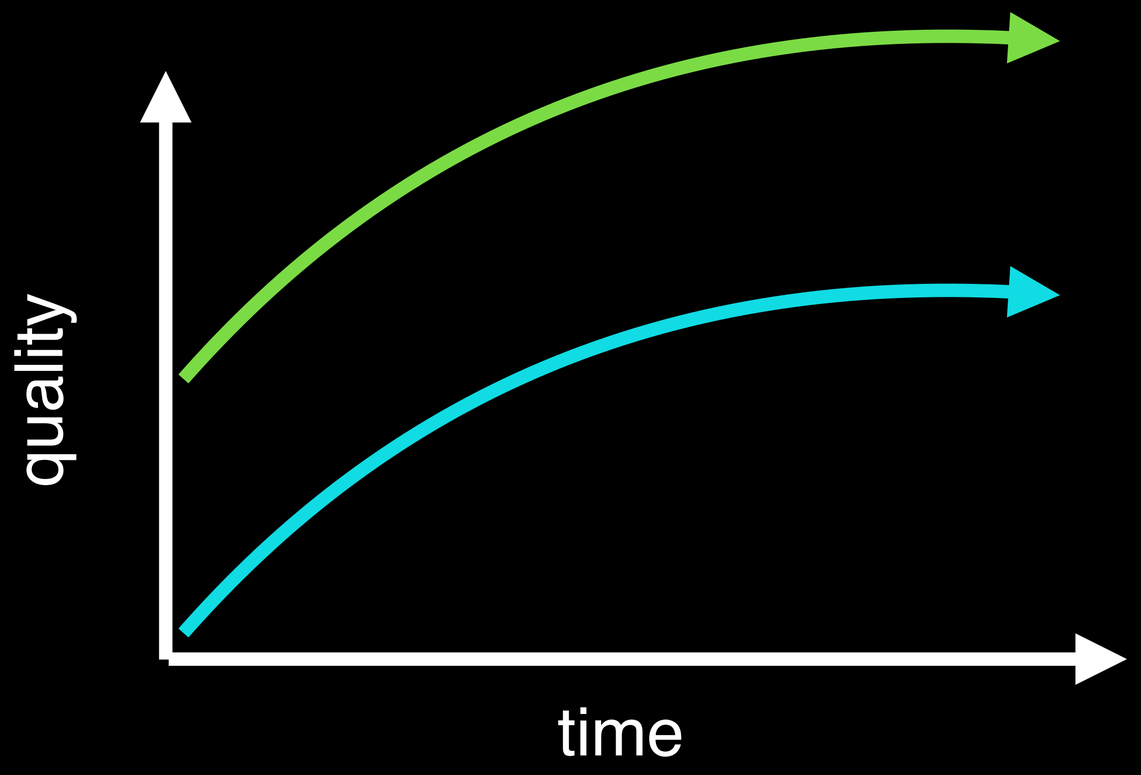
compare redux and alt:

- concepts
- testing
- trends

agenda

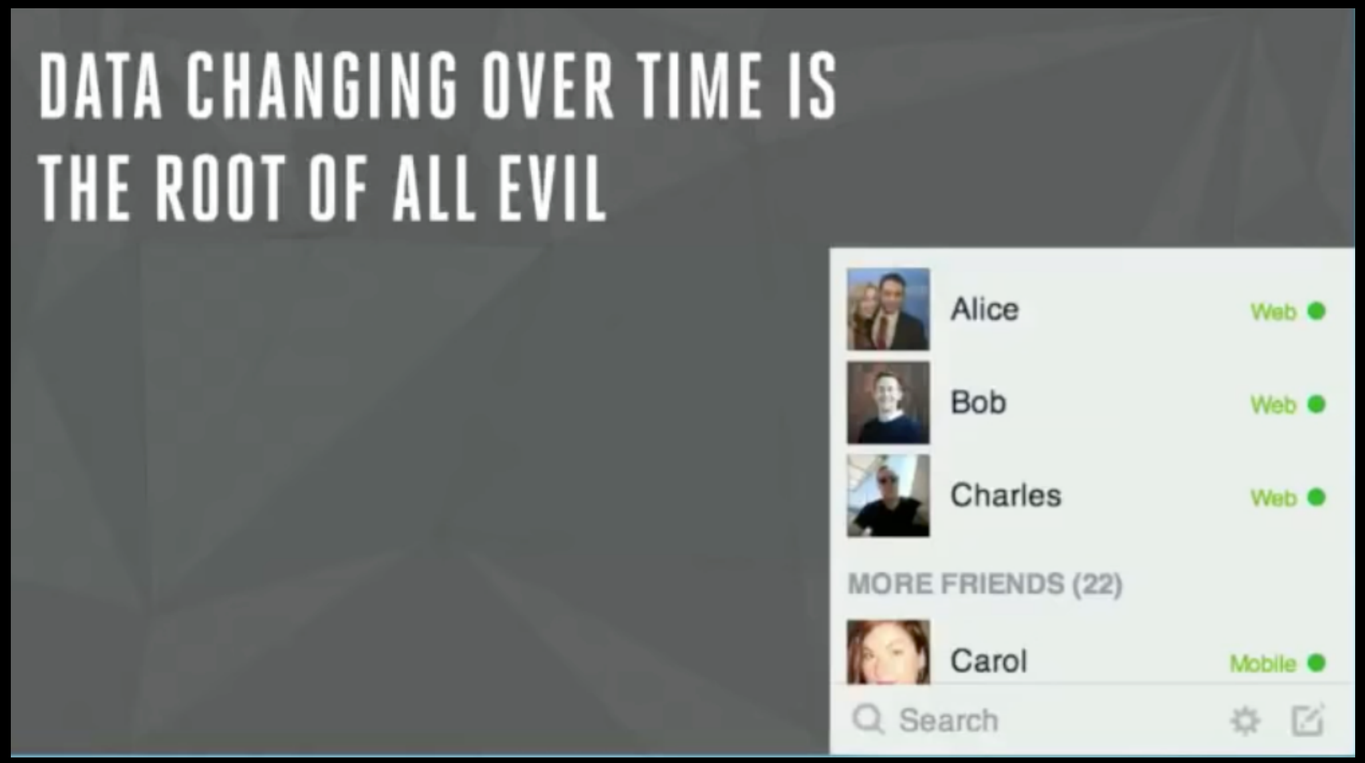
- why?
- react
- flux
- redux and alt comparison
- microservices for client side

why?



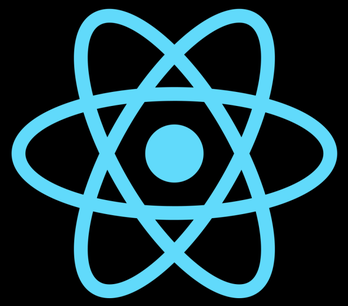
quality takes time

fb contact list



react

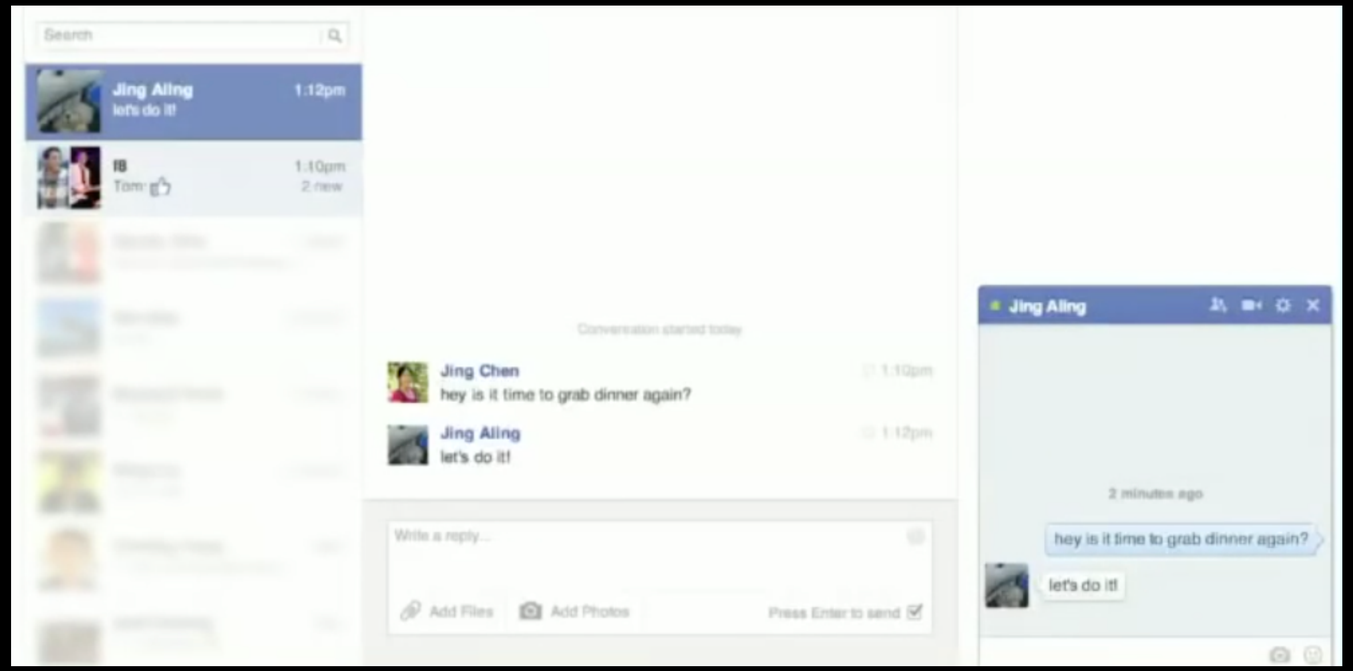
- just the UI
- virtual DOM
- one way data flow



```
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

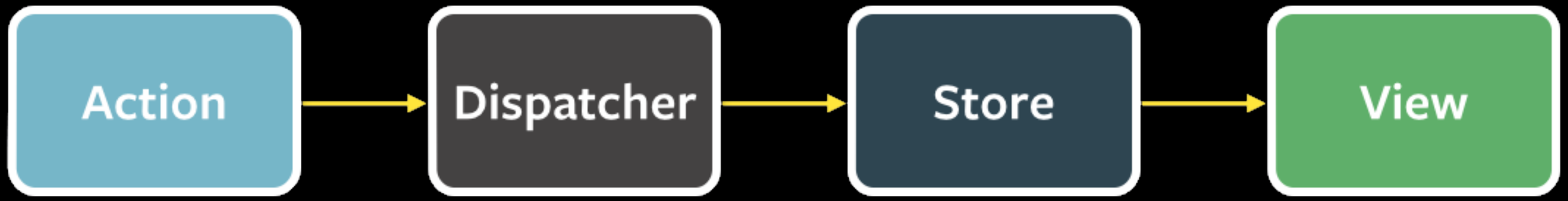
Hello John

fb chat


Search
Jing Aling
let's do it!
1:12pm
FB
Tom
1:10pm
2 new
Conversation started today
Jing Chen
hey is it time to grab dinner again?
1:10pm
Jing Aling
let's do it!
1:12pm
Write a reply...
Add Files
Add Photos
Press Enter to send
Jing Aling
2 minutes ago
hey is it time to grab dinner again?
let's do it!


1

flux



Flux is the application architecture that Facebook uses for building client-side web applications. It complements React and is more of a pattern than a framework.

redux and alt

- implementations of flux
- isomorphic - (equal shape when rendered both on client and server side)
- support react-native

redux

predictable state container for JavaScript apps

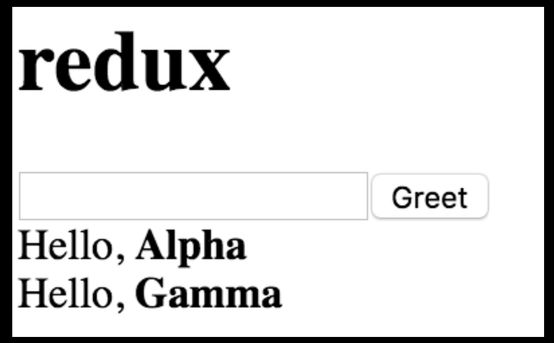
- whole state of app is stored in an object tree inside a single store
- only way to change the state tree is to emit an action, an object describing what happened
- to specify how the actions transform the state tree, you write pure reducers
- great documentation - not that it needs one really

alt

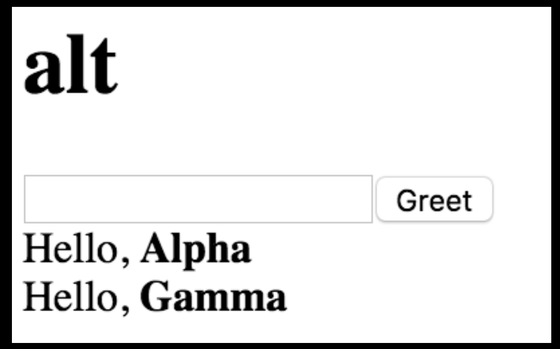
Isomorphic flux implementation

- pure flux
- **actively** maintained
- extremely flexible and **unopinionated** in how you use flux
- has logo
- lacks up to date documentation and samples

hello world




```
├── app.jsx
├── components
|   ├── greet.jsx
|   ├── hello.jsx
|   ├── hello.test.js
|   └── helloList.jsx
├── index.html
├── index.js
├── state
|   ├── namesReducer.js
|   ├── namesReducer.test.js
|   └── store.js
```




```
├── actions
|   └── NamesActions.js
├── alt.js
├── components
|   ├── greet.jsx
|   ├── hello.jsx
|   ├── hello.test.js
|   └── helloList.jsx
├── index.html
├── index.js
├── state
|   ├── configureNamesStore.js
|   ├── configureNamesStore.test.js
|   └── namesStore.js
```

components

- similar or equal setup for tiny components
- differs for larger components that contain actions and more logic
- testing is easy using react test utils or wrapper libraries
- redux - allows function syntax, alt - can get messy when static functions needed

components - code

```
import React from 'react';

const Hello = ({name}) => {
  return (
    <div>
      Hello, <b class='test-name'>{name}</b>
    </div>
  );
};

export default Hello;
```

components - tests

```
describe('hello component', () => {
  it('renders name', () => {
    const output = renderHello('Steve');
    const name = getChildrenByClass(output, 'test-name');
    expect(name.props.children).toEqual('Steve');
  });
  it('renders component', () => {
    const output = renderHello('Steve');
    const expected =
      <div>Hello, <b class="test-name">Steve</b></div>;
    expect(output).toEqualJSX(expected);
  });
});


const renderHello = name => {
  const renderer = TestUtils.createRenderer();
  renderer.render(<Hello name={name}/>);
  return renderer.getRenderOutput();
}
```

components - tests

```
describe('<MyComponent />', () => {

  it('renders three <Foo /> components', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.find(Foo)).to.have.length(3);
  });

  it('renders an `.icon-star`', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.find('.icon-star')).to.have.length(1);
  });

  it('renders children when passed in', () => {
    const wrapper = shallow(
      <MyComponent>
        <div className="unique" />
      </MyComponent>
    );
    expect(wrapper.contains(<div className="unique" />)).to.equal(true);
  });

  it('simulates click events', () => {
    const onButtonClick = sinon.spy();
    const wrapper = shallow(
      <Foo onButtonClick={onButtonClick} />
    );
    wrapper.find('button').simulate('click');
    expect(onButtonClick.calledOnce).to.equal(true);
  });
});
```

airbnb/enzyme

react element

css class

html element

simulate event

stores

- redux - one, alt - many
- redux - registers reducers, alt - defines logic to execute based on data passed by action
- redux - no tests needed, test reducers, alt - the meat of logic, harder to test without side effects
- redux - immutable, alt - mutable
- alt - ReferenceError: A store named NamesStore already exists, double check your store names or pass in your own custom identifier for each store]

stores - code

```
import alt from '../alt';
import namesActions from '../actions/NamesActions';

class NamesStore {
  constructor() {
    this.bindListeners({ add: namesActions.ADD})
    this.state = {
      names: []
    }
  }
  add(name) {
    if (name === 'Stranger') {
      return this.state;
    }
    this.setState({ names: [...this.state.names, name] });
  }
}

export default function configureStore() {
  return alt.createStore(NamesStore, 'NamesStore');
}
```

stores - tests

```
import alt from '../alt';
import configureStore from './configureNamesStore';
import namesActions from '../actions/NamesActions';

const greet = name => {
  const action = namesActions.ADD;
  const data = name;
  alt.dispatcher.dispatch({ action, data });
};

describe('store', () => {
  it('starts off with empty names array', () => {
    const store = configureStore();
    const state = store.getState();
    expect(state).toEqual({ names: []});
  });
  it('adds a name after greet action processed', () => {
    const store = configureStore();
    greet('Steve');
    const state = store.getState();
    expect(state).toEqual({ names: ['Steve']});
  });
});
```

reducers

- pure functions - zero side-effects, only rely on parameter values
- super easy to test
- they scale - easy to add a new one

reducers - code

```
const shouldGreet = name => {
  return name !== 'Stranger';
};

const namesReducer = (state = { names: [] },  action) => {
  if (action.type === 'GREET' && shouldGreet(action.payload.name)) {
    return { names: [...state.names, action.payload.name] };
  }
  return state;
};

export default namesReducer;
```

reducers - tests

```
import expect from 'expect';
import namesReducer from './namesReducer';

const greet = name => {
  return { type: 'GREET', payload: { name: name }};
};

describe('names reducer', () => {
  it('greets', () => {
    const state = namesReducer({ names: [] }, greet('Steve'));
    expect(state).toEqual({ names: ['Steve']});
  });
  it('no greetings to Stranger', () => {
    const state = namesReducer({ names: [] }, greet('Stranger'));
    expect(state).toEqual({ names: []});
  });
});
```

react-redux glue

```
const store = configureStore();

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('App')
)
```

1

react-redux glue

```
import React from 'react';
import {connect} from 'react-redux';
import Hello from './hello';

const HelloList = ({ names }) => {
  return (<div className='test-hello-name-list'>
    {names.map(name =>
      <Hello name={name}/>
    )}
  </div>);
}

const mapStateToProps = state => {
  return {
    names: state.names
  }
}

export default connect(mapStateToProps)(HelloList);
```

2

react-alt glue

```
// Bad: getting into lib internals
import connectToStores from 'alt-utils/lib/connectToStores';

class HelloList extends React.Component {
  static getStores(props) {
    return [store];
  }
  static getPropsFromStores(props) {
    return store.getState();
  }
  render() {
    return (
      <div className='test-hello-name-list'>
        {this.props.names.map(name =>
          <Hello name={name}/>
        ) }
      </div>
    );
  }
}

export default connectToStores(HelloList);
```

1

react-alt glue

```
class Greet extends React.Component {
  constructor(props) {
    super(props);
    this.submit = this.submit.bind(this);
  }
  static getStores(props) {
    return [store]
  }
  static getPropsFromStores(props) {
    return store.getState();
  }
  submit(e) {
    e.preventDefault();
    namesActions.add(this.input.value);
  }
  render() {
    return (
      <div className='test-greet'>
        <form onSubmit={this.submit}>
          <input id='test-greeting' type='text' ref={node => {
          this.input = node}}/>
          <button id='test-submit-greeting' type='submit'>Greet</button>
        </form>
      </div>
    );
  }
};

export default connectToStores(Greet);
```

1

application stats

lines of code

21 ./hello-redux/components/greet.jsx
11 ./hello-redux/components/hello.jsx
32 ./hello-redux/components/hello.test.js
12 ./hello-redux/components/helloList.jsx

6 ./hello-redux/state/store.js

12 ./hello-redux/state/namesReducer.js
17 ./hello-redux/state/namesReducer.test.js

19 ./hello-redux/app.jsx
14 ./hello-redux/index.js

144 total

35 ./hello-alt/components/greet.jsx
11 ./hello-alt/components/hello.jsx
32 ./hello-alt/components/hello.test.js
25 ./hello-alt/components/helloList.jsx

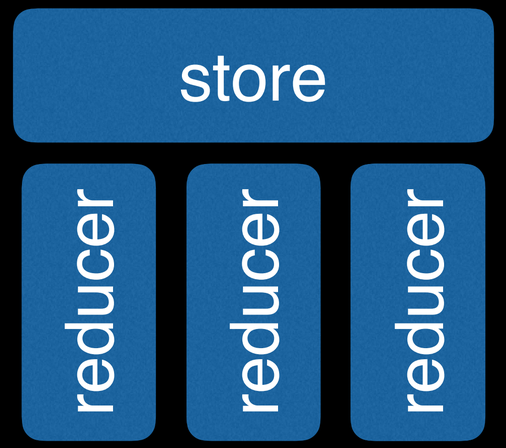
20 ./hello-alt/state/configureNamesStore.js
24 ./hello-alt/state/configureNamesStore.test.js
2 ./hello-alt/state/namesStore.js

8 ./hello-alt/actions/NamesActions.js
4 ./hello-alt/alt.js
12 ./hello-alt/index.js

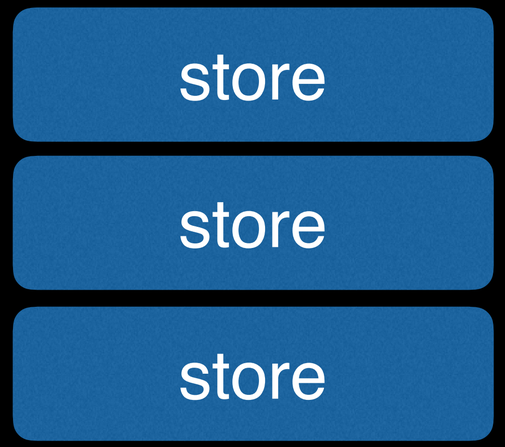
173 total = 20 % more

scaling application

react



alt



stats

	redux	alt
version	3.5.2	0.18.4
size	5.9K (2kb?)	23K
downloads / month	670K	38K
repo	reactjs/redux	goatslacker/alt
open issues	44	79
pull requests	20	4
last commit	hours ago	almost a month ago

comparison

redux

alt

Open Systems Don't Always Win - Steve Jobs

verdict

TRIAL

Redux is a great, mature tool that has helped many of our teams reframe how they think about managing state in client-side apps. Using a Flux-style approach, it enables a loosely coupled state-machine architecture that's easy to reason about. We've found it a good companion to some of our favored JavaScript frameworks, such as Ember and React - ThoughtWorks TechRadar 2016 April

HOLD

Alt although says to be unopinionated it pushes to use action objects, static functions that are used to connect components to stores, possible to mutate store state and having multiple stores might encourage a monolith approach, doesn’t have as big community backing as Redux and feels dated in comparison

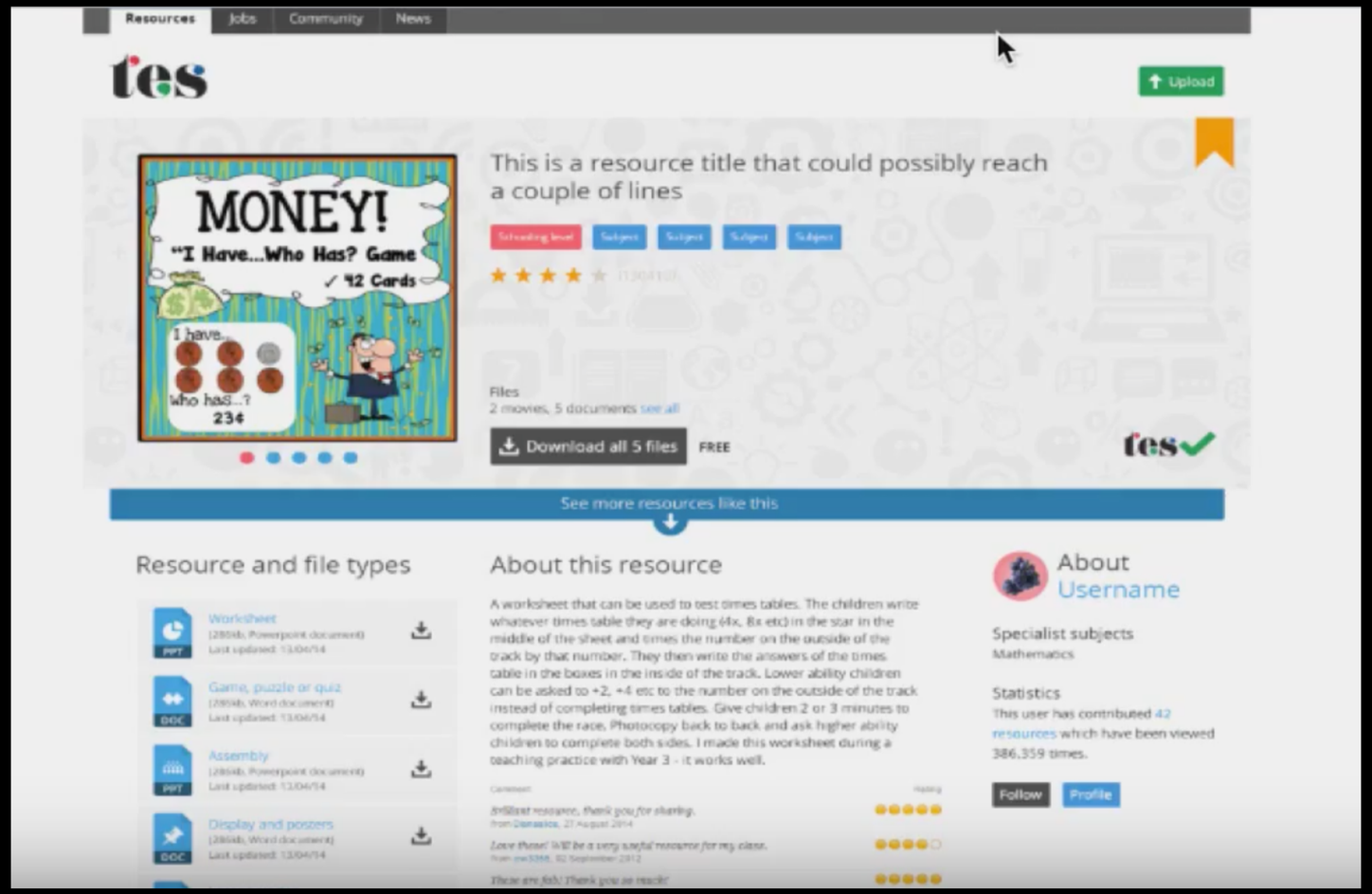
end to end tests

```
var process = require('process');
var port = process.env.PORT;

function navigateToSite(test) {
  return test.open('http://localhost:' + port);
}

module.exports = {
  'says hello': function (test) {
    navigateToSite(test)
    .assert.doesntExist('.test-name', 'initial state has no greetings')
    .type('#test-greeting', 'Steve')
    .click('#test-submit-greeting')
    .assert.text('.test-hello', 'Hello, Steve', 'adds a new greeting')
    .done();
  }
};
```

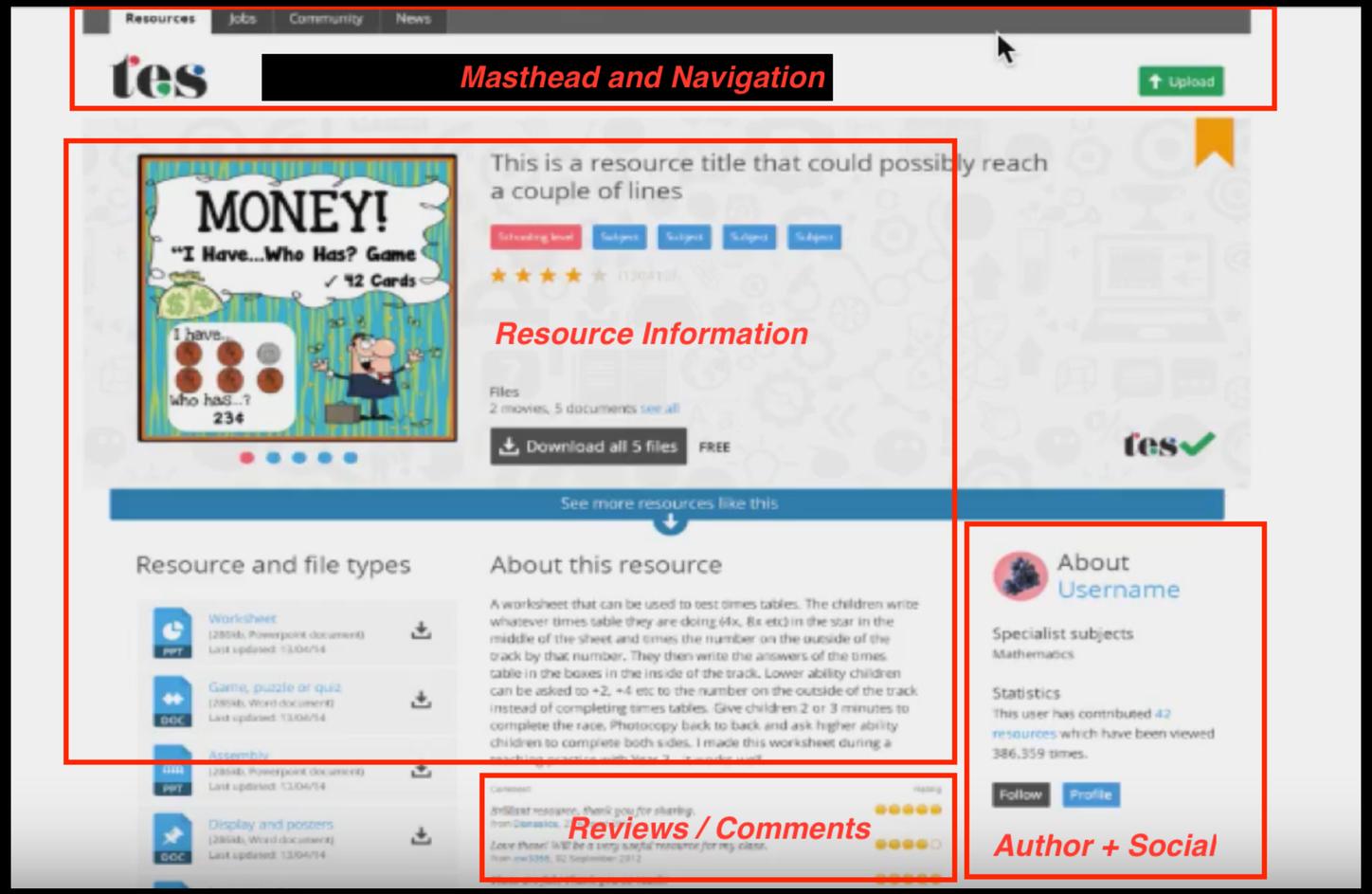
give confidence when switching frameworks

*don’t use dalekjs

example site - TES Global

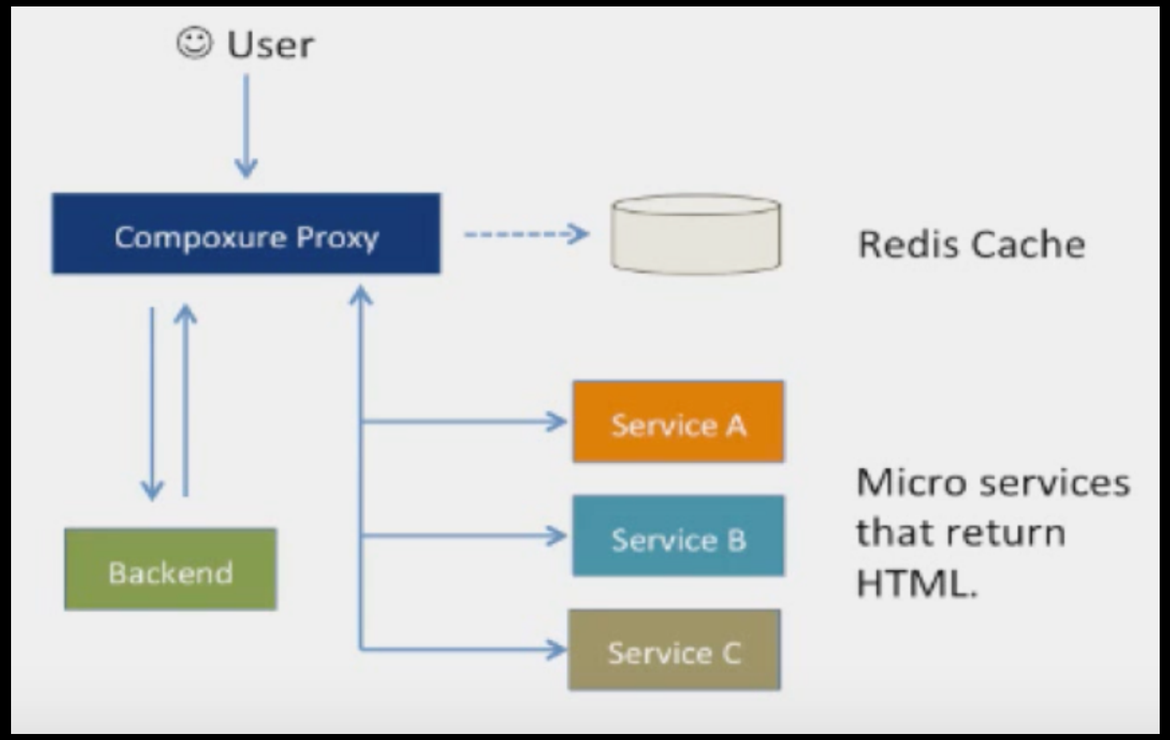


microservices



https://www.youtube.com/watch?v=_SoPZS7Wqiw

one way to implement



implementation by TES Global

q&a

It seems that perfection is attained, not when there is nothing more to add, but when there is nothing more to take away

– Antoine de Saint Exupéry

links

- http://redux.js.org
- http://alt.js.org
- https://github.com/airbnb/enzyme
- https://egghead.io/series/react-testing-cookbook
- https://egghead.io/series/getting-started-with-redux
- http://redux.js.org/docs/recipes/WritingTests.html
- https://github.com/gaearon/redux-devtools
- https://github.com/uldissturms/talks